The Nature of evidence in empirical software engineering

Conference or Workshop Item

The Nature of Evidence in Empirical Software Engineering

Judith Segal
Department of Computing, Faculty of Mathematics and Computing, The Open University, Walton Hall, Milton Keynes MK7 6AA, UK
j.a.segal@open.ac.uk

Abstract

In this paper, we argue that the gap between empirical software engineering and software engineering practice might be lessened if more attention were paid to two important aspects of evidence. The first is that evidence from case or field studies of actual software engineering practice is essential in order to understand and inform that practice. The second is that the nature of evidence should fit the purpose to which the evidence is going to be put. One type of evidence is not per se better than another. For example, the evidence required to persuade a manager to change an aspect of practice might be totally different in nature from that required to deepen the academic community's understanding of such practice.

1. Introduction

The empirical software engineering community seeks to conduct empirical studies so that the practice of software engineering might be informed by the evidence resulting from these studies, similar to the way that the practice of conventional engineering disciplines is informed by the laws of science. As Shepperd says in [1]:

'… systematic empirically based research is essential in order to move software engineering on from an advocacy based discipline to an evidence based discipline' [1: p.37]

It is clear, however, that this aspiration has not yet been achieved. Zelkowitz et al., [2] put the situation starkly:

'Clearly the research community is not generating results that are in tune with what industry needs to hear, and industry is making decisions without the benefit of good scientific developments. The two communities are severely out of touch with one another'. [2: p.231]

One has to consider why the empirical software engineering community is apparently having such little effect on practice. Is it simply the fact that software engineering hasn't been practised for long enough to accrue the laws, techniques, assumptions and heuristics which inform the practice of other engineering disciplines? Or is it the fact, as suggested by the quote from [2] above and also in [3], that empirical studies often fail either to address the concerns of practitioners or to produce results which are directly usable by them? That is, is the evidence produced by such studies, of the wrong type, in some sense?

In this paper, we explore two partial answers to this latter question. Firstly, we assert that the evidence produced from empirical studies of software engineering *practice* is essential in order to deepen our understanding of such practice and hence to devise studies which are directly relevant to practitioners. Furthermore, we believe that not enough emphasis has been placed on this type of study in the past. Secondly, we argue that empirical software engineers should be aware of the need to match evidence with its intended purpose: evidence which satisfies academic criteria may be different from that which persuades practitioners and their managers to participate in studies or to apply the results thereof to their own practice. Zelkowitz et al. begin to explore this latter point in [2]:

'The research community is more concerned with theory confirmation and validity of the experiment and less concerned about costs, whereas the industrial community is more concerned about costs and applicability in their own environment...' [2: p.256]

As we shall discuss in section 5 below, we believe that in the field of empirical software engineering, quantitative evidence has historically been valued over qualitative. For example, Lanubile argues in [4] that

Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'04)
0-7695-2218-1/04 $20.00 © 2004 IEEE

'Empirical [software engineering] research assumes that all constructs of interest must have observable features that we can measure, although imperfectly' [4: p.98]

More recently, Seaman, has argued cogently in [5] that quantitative studies alone give an impoverished view of software engineering and that

'In software engineering, the blend of technical and human behavioral aspects lends itself to combining qualitative and quantitative methods, in order to take advantage of the strengths of both' [5: p.577]

Petre, as quoted in [6], argues that the research question should determine the nature of the evidence required. She asserts that the important point is to determine the research question, then to determine the type of evidence that will answer the question, and finally, to determine the research method which will deliver the right sort of evidence.

In this paper, we go beyond Petre's position to suggest that it is not just the research question which determines the type of evidence, but also the intended audience for the evidence, that is, the people whom the evidence is intended to persuade. To reiterate the point: we take for granted herein the fact that, in the field of empirical software engineering, one type of evidence is not per se better than another. Rather, the decision as to which type of data should be collected and how it might be analysed, interpreted and presented to form evidence, depends on the use to which the evidence is going to be put.

We structure this paper by means of the following simple taxonomy of the possible aims of an empirical study of software engineering: An aim might seek to promote

- the development of an understanding of the nature of software engineering practice. (We argue in section 2 below that this should be a central aim of empirical software engineering.) ;
- an exploration of how this practice might be improved;
- an evaluation of the effects of introducing a conjectured improvement into practice.

In sections 2 to 4 below, we shall explore for each aim, the type of evidence which will further the aim, and the type of study which will produce this evidence. We shall also, where relevant, explore the different types of evidence required for different types of audience. In section 5, we shall discuss the results of our exploration.

2. The aim: the development of an understanding of the nature of software engineering practice

In this section, we shall firstly defend our position that this aim should be central to empirical software engineering, and then consider the nature of the evidence required in support of this aim.

2.1. The importance of this aim: study of practice versus laboratory experiments

We alluded above to the concern expressed, for example, in [2] and [3], about the gap between empirical software engineering and practice. This gap is not unique to empirical software engineering: Glass et al. in [7] describe a survey of over 300 papers in software engineering over a period of 5 years and conclude that

'There is a severe decoupling between research in the computing field and the state of the practice in the field' [7: p.505]

Given this concern, it is surprising to us that developing an understanding of the nature of practice does not enjoy a more central position within empirical software engineering.

Indeed, some empirical software engineers appear to believe that the nature of software engineering practice – what actually happens in the real world when software is developed – is unimportant and that experiments within a laboratory suffice. For example, Munch et al. [8] claim that:

'Laboratory experiments have been proved as an essential means for determining the effects of software engineering technologies'

and Kitchenham, Linkman and Fry observe in [9] that

'Many software experiments take place in a classroom setting often in the context of an examination or test' [9: p.12]

The major problem with laboratory experiments in the software engineering context is that they factor out the human dimension, how people actually work in an actual context, and hence ignore what Seaman describes in [5] as

'the central role of human behavior in software development' [5: p.577]

Given this central role, we should not be surprised that traditional scientific laboratory experiments, designed to test hypotheses by the use of dependent and independent variables, experimental and control groups, and statistical analyses of the resultant quantitative data, are difficult to carry out in the context of empirical software engineering. Kitchenham et al. in [10] and Segal in [11] discuss some of these difficulties, such as the identification, precise definition and interdependence of relevant variables. We should also not be surprised that, even when such an experiment is carried out to an acceptable standard of scientific rigour, its results might fail either to transfer to an actual development environment or to convince practitioners, as Zelkowitz et al. note in [2]:

'… the industrial community is generally wary of laboratory research results…' [2: p.255]

Kitchenham, Linkman and Fry in [9] identify some problems (situational effects) with laboratory experiments: treatments on small tasks may not scale up to realistically-sized tasks, whereas treatments which have overheads (for example, of learning) making them unsuitable for laboratory-sized tasks, may be eminently suitable for larger tasks; people in a real-world setting can take advantage of informal knowledge networks, and so on. They conclude that, despite the prevalence of laboratory studies in empirical software engineering,

'…. in our view, industrial case studies may be the best way to address the situational effects likely in software engineering studies' [9: p.12]

We should note that concern about the effectiveness of controlled laboratory experiments in closing the gap between research and practice is not confined to empirical software engineering. For example, Rogers makes the following point about human-computer interaction in [12]:

'The stark differences between a controlled lab setting and the messy real world setting, [means] that many of the theories derived from the former [are] not applicable to the latter'

We believe this point to be just as relevant to software engineering as to human computer interaction.

So far in this section, we have argued that understanding the nature of software engineering practice should be central to the discipline of empirical software engineering, since it is only through this understanding that studies can be devised which produce evidence which is both relevant to, and usable by, practitioners. We have also argued that laboratory experiments are a poor substitute for a study of practice *if* the aim is to understand that practice. However, we recognise that studies of practice might be both difficult and resource intensive to carry out; practitioners and their managers may be loath to expose the details of their practice to external investigators, and the gains from such a study may not repay the resources expended. In such cases, laboratory studies may be all that is possible. In addition, a laboratory study may also act as a valuable precursor to a study of practice in vivo, by providing evidence which persuades practitioners and their managers to permit such a study.

We now turn our attention to the nature of the evidence which might enhance our understanding of practice.

2.2. The nature of the evidence which might further this aim

Here, we shall firstly discuss how the underlying data might be collected, and then how it might be analysed, interpreted and presented to form evidence.

We believe that data relevant to pursuing this aim can best be collected by studying practice itself as a richly contextualised whole. This may be achieved by means of ethnographically based studies. Such a study may be either a field or an in-depth case-study: the difference between the two is not clear-cut though the former may involve a greater length of time and degree of immersion than the latter, see, for example, Klein and Myers in [13]. Pure ethnography is often impractical, involving as it does, immersion in the domain of study over a long period of time and collection of a plethora of data, with no item of data being given primacy over any other. In [14], we suggest ways in which ethnography might be modified in order to fit the purposes of empirical software engineering.

Any ethnographically-based study must adhere to two basic principles: the data must occur naturally as people go about their ordinary business (that is, it may consist of videos of work meetings, or of the artefacts of work, such as documents or code, but it does NOT occur as the result of some intervention by the investigator, as in traditional scientific experiments), and the investigator must be detached enough from the situation under investigation to be able to identify and question taken-for-granted knowledge and tacit assumptions, values, beliefs – what is referred to by Cook and Brown in [15] as 'tacit group knowledge'. In addition to these two basic principles, and cognisant of the fact that there is no such thing as "pure" observation – what any observer sees in a particular situation and how s/he interprets what s/he has seen depends on her/his experience, expectations, mental model of the situation, see, for example, Klein and Myers in [13] and

Chalmers discussing the philosophy of science in [16] – we strongly advise that the investigator/observer has enough understanding of software engineering to know what is, and is not, likely to be significant. For example, someone with some understanding of the context will recognise that the fact that a particular software engineer wears a yellow jumper is unlikely to be significant, whereas the fact that her/his machine is covered by aide-memoires might well be. A pure ethnographer might give equal status to both facts.

If ethnographically based studies are impracticable, then other qualitative methods might be used in order to probe the nature of practice. Such methods include interviews conducted by the researcher with the software engineer, think-aloud and retrospective protocols. None of these are ethnographic methods in that they require intervention from the researcher or place constraints on normal ways of working. Of course, in her/his everyday practice, the software engineer might be involved in interviews (with other software engineers or with customers, for example), or might talk aloud as s/he follows a procedure (for example, if s/he habitually talks aloud, or if s/he is explaining what s/he is doing to a colleague). In this case, the data which arise are truly ethnographic.

With non-ethnographic qualitative methods, it is important to confirm one's findings explicitly through 'triangulation', that is, data from more than one source. Data from a single source (for example, from interviews or from documents or from protocols) can be suspect, as is discussed in Bratthall and Jorgensen, [17]. For example, the cognitive effort required to think aloud while performing a task might effect how the task is undertaken; in retrospective protocols, a software engineer might forget why s/he did what s/he did at a particular point; finally, with the best will in the world, software engineers might not give truthful answers to survey or interview questions because their perceptions may not match with the actuality.

Examples of software engineers' perceptions not matching with the reality captured by observational, ethnographically based studies, include the following. Jorgensen, as cited in [17], describes a situation in which the time software engineers claimed to spend on fault correction, was about twice that actually observed. Singer et al. [18] found that although the software engineers observed *said* they focussed on documentation, in fact, they mostly searched and read source code. Finally, Visser et al., [19], found that, at a time when 'top-down software design' was de rigueur and assumed without question to be the way that expert designers worked, in fact, observations showed that many such experts worked in a far less structured, more opportunistic way.

For these reasons – the effect that the research method has on the work process (for example, the non-natural production of talk-aloud data); the frailty of human memory in retrospectives; the possible gap between perception and reality – and also because ethnographically based studies can reveal the taken-for-granted and the tacit (as discussed in [14]), we feel that ethnographically-based studies provide richer information about the reality of practice than non-ethnographic methods. In ethnographically-based studies, triangulation is not an issue: ethnographic data are likely to be a mixture of documents, transcripts, notes of investigator observations, photos, work products, and, perhaps, some quantitative data (for example, the average length and frequency of particular meetings; how frequently someone has consulted a reference).

Having discussed how the data might be collected, we now consider how they might be analysed, interpreted and used. This depends on the use to which the data are going to be put.

Seaman in [5] and Lee in [20] both discuss how field study/case study data might support a natural science view of software engineering (Seaman) and information systems (Lee) Seaman describes how such data might be analysed so as to provide hypotheses/conjectures and Lee discusses how such hypotheses can be confirmed or disconfirmed using naturally occurring events and controls. In Lee's exemplar case-study, the hypothesis confirmed is that resistance to the implementation of a particular information system is related to the way that the system effects the distribution of power within the organisation; the hypotheses disconfirmed are that this resistance is related to technical inadequacies in the system or to individual human factors.

Klein and Myers in [13] take an interpretivist rather than a natural science view, and discuss some principles by which case studies might be conducted in order to provide plausible and cogent accounts of the situation under investigation. These principles include:

- the need to reflect on the context (social and historical) of the situation;
- the recognition of the effect of interactions between the researcher and the participants;
- the fact that understanding of parts both informs and is informed by understanding of the whole;
- the necessity for multiple interpretations of the same data.

Interpretivist accounts may both enhance understanding and persuade practitioners to try something new. Richly contextual case studies, where practitioners can recognise a situation similar to their own, might evoke the gut instinct 'it worked there; it

will work here. I must try it'. For example, we speculate that the rapid growth in agile methodologies is not due to the presentation of any hard empirical evidence (of which there is currently not very much), but to practitioners' gut reactions on hearing of other's experiences: 'yes, this makes sense. This is how it should be done'. One interpretivist study which provides a rich account of eXtreme programming practice, is that of Robinson and Sharp [21].

To summarise this section: we have argued that studies of practice are essential both to spur conjectures/hypotheses as to how such practice might be improved, and to provide cogent and plausible accounts of such practice. Such accounts will not only serve to deepen our understanding of the nature of practice but also might persuade practitioners to reflect on, and try out conjectured improvements in, their practice. We have also argued that, where possible, ethnographically-based studies are best suited to elucidate practice, and have described some studies in which assumptions or perceptions have been confounded by observations.

3. The aim: exploring how practice might be improved

As we have noted above, Seaman in [5] discusses how evidence from ethnographically-based studies of practice can be used to spur conjectures/hypotheses about how such practice might be improved. However, these studies are not the only source of suggestions as to how practice might be improved. Such evidence may also come from:

- Traditional empirical software engineering sources, such as researchers' reflections and ideas, tested in the laboratory;
- workshops or other practitioner gatherings;
- individual practitioners reflecting on their practice;
- studies of the software product.

As an example of the last-named, Cartwright and Shepperd in [22] describe the study of a large object-oriented code-base in order to investigate the distribution of defects, and identified classes that were part of inheritance structures as being particularly error-prone. This led to the writers making two suggestions for the improvement of practice: firstly to argue for concentrating software verification effort on these classes, and secondly, to suggest that developers consider seriously for each particular project whether inheritance is necessary.

Having produced conjectures/hypotheses as to how the practice of software engineering might be improved, we suggest that the empirical software engineer should now consider how the conjectured improvement, the intervention, might be introduced into practice. Previous studies on effecting anything other than minor changes in software engineering practice, for example, the introduction of a software process improvement or of a reuse program, have stressed the need for management support, see for example, Curtis [23] and Hall and Wilson [24] looking at software process improvement, and Morisio et al. in [25], looking at reuse. In addition, we speculate that change in practice cannot be sustained over a period of time without support from the practitioners themselves. So we believe that the empirical software engineer, having convinced himself/herself of the value of a particular intervention, should now determine whether the evidence which convinced him/her might also suffice to persuade management and practitioners to adopt the intervention, or whether further evidence is necessary.

In order to convince management that it is worth the investment necessary to introduce the intervention, it might be desirable to appeal to financial considerations. Ideally, we should like to be able to say 'we have evidence that if you institute this particular tool/technique/method, then you will save money. We have found that adopting this intervention means your percentage of code defects will reduce *or* your production rate will increase *or* you will be able to produce a certificate which will make it easier to attract customers *or*…..' We are now in the world of metrics, and the quantitative evidence supporting our assertions may come from laboratory experiments.

As to convincing practitioners, we argued in 2 above that practitioners might be persuaded by cogent plausible accounts afforded by case-studies of contexts which they recognise. They may also be persuaded by the highly unscientific, unreliable but sometimes compulsively persuasive, evidence provided by peer anecdotes or by vendors, as recognised in [2]. Interpretivist case studies, for example, the results of ethnographic studies, differ significantly from anecdote in aiming to provide a rounded, unbiased account, by adhering to the kind of principles described in 2.2. above, such as seeking multiple interpretations of the same phenomenon, and being aware of how data is filtered and interpreted through interactions between the researcher and the subject. See Klein and Myers, [13], for details.

4. The aim: an evaluation of the effects of introducing a conjectured improvement into practice

Here we wish to ask questions such as:

1. has the intervention improved practice or the product in any tangible, quantitative way?
2. has the intervention improved practice or the product in some less tangible way?
3. is the intervention sustainable? That is, is the intervention fully embedded as part of practice?

In order to answer question 1, we are back in the world of metrics and comparing the value of some quantitative measure taken before and after the intervention was introduced, as in the standard pattern for software process improvement.

As to question 2, Kitchenham et al. in [10] warn against concentrating on quantitative data at the expense of qualitative, using as an example a number of quantitative studies which suggested that inspection meetings are not necessary to maximise defect detection, but ignored the qualitative benefits of such meetings, such as on-job training and the promotion of teamwork.

Boehm and Turner in [26] stress the importance of question 3, noting that one shouldn't get carried away by the initial euphoria of early adopters. In their discussion of how agile and plan-based methods might be melded, they note that the C3 project, which acted as a catalyst for the development of eXtreme Programming and which was initially very successful, was eventually cancelled. The adoption and diffusion throughout an organisation of new technologies and procedures continues to be an active topic of research in the Information Systems and Business Management communities.

Answers to questions 2 and 3 can best be sought, we believe, through field or extended case-studies, as described in section 2 above.

5. Discussion and conclusions

This paper has been concerned with two issues: the importance of studies of software engineering practice, and the nature of evidence.

We have argued that without understanding the nature of practice, empirical software engineers will find it difficult to identify and promote potential improvements in practice. We have suggested that ethnographically-based studies provide the richest picture of practice, while accepting that they might sometimes be impracticable, and that other qualitative methods, suitably triangulated, should then be used.

As to the nature of evidence, we have argued here that what constitutes evidence depends on the purpose to which the evidence is going to be put. For example, if one wants to persuade an executive manager that his workforce should adopt a particular practice/tool/method, one appeals to hard evidence, produced, perhaps, within a natural science experimental framework, and presented in terms of financial considerations. However there is plenty of evidence in the literature of executive management being influenced by social, political or organisational pressures, and making decisions on the basis of, for example, 'jumping on the bandwagon' rather than on the basis of any hard evidence, see, for example, Hall and Wilson [24], discussing the making of decisions relating to software process improvement initiatives, and Waterson et al. [27], considering decision making about the use of CASE tools and outsourcing. If one wants to ascertain why executive management doesn't make a particular decision in the face of seemingly compelling hard evidence, then one might appeal to the qualitative evidence produced by an ethnographically-based field study.

We argue that, in empirical software engineering, there is no 'pyramid of data' (a phrase we have heard in the context of evidence-based medicine). That is, one type of data does not per se have primacy over another: quantitative data is not necessarily superior to qualitative.

Seaman in [5] describes qualitative data as having "an image problem" in the context of empirical software engineering. Perhaps the cause of this image problem is the widespread view that, firstly, only quantitative data is purely objective in truly reflecting the world without investigator bias, and secondly, that only quantitative data contributes to hard science, and that empirical software engineers should aspire to be hard scientists.

Regarding the first point, Lanubile in [4] asserts that objectivity is independent of whether data are quantitative or qualitative.

'The objectivity of empirical research comes from a review process that assures that the analysis relies on all the relevant evidence and takes into account all the rival interpretations. This can be done (or not done) in both a quantitative and a qualitative analysis'. [4: p.102]

In [9], Kitchenham, Linkman and Fry point out several potential sources of bias in quantitative data. For example, there is potential bias involved in measurement (measurements may be rounded to fit in with the investigator's model; bias might be introduced unwittingly through deficiencies in the measuring instrument), to say nothing of the potential bias in applying statistical tests (for example, the identification by the investigator of outlier data, to be ignored, depends on the investigator's subjective expectation of what the data should be like). We have already discussed how bias might be mitigated in

ethnographically-based case studies by consideration of multiple interpretations and the like.

There is also the fact that not all aspects of the world are quantifiable: Bannon, as quoted in [6], describes the ignoring of aspects which can't be measured, as "suicide".

With regard to the relationship between objectivity, hard science and software engineering, philosophers of science question just how objective 'hard science' really is (see, for example, Chalmers in [16]). Some philosophers contend that the validity of observations, theories and hypotheses in a particular scientific culture at a particular point in time depends on the consensual agreement of the members of the scientific community at that time and in that culture, rather than on any hard and fast evidence. Finally, we agree with Seaman in [5] that software engineering is not 'hard science', in the sense that it is not independent of contextual factors such as that provided by the human dimension.

We should reiterate that, in challenging the perceived primacy, historically, of quantitative over qualitative data in empirical software engineering, we are not arguing for the reverse primacy, of qualitative over quantitative. Rather, we argue that hard quantitative data is not, of itself, better than soft qualitative data, and neither is qualitative data always better than quantitative. Different purposes demand different sorts of data, and different sorts of analysis, interpretation and representation of that data to provide evidence.

We have argued in this paper that the purpose of understanding the practice of software engineering should be central to empirical software engineering, and that it is not enough for an investigator to convince herself/himself of the efficacy of some change in practice, s/he must also convince practitioners and their managers, and this might demand quite different types of evidence from that which convinced the investigator and his/her community. Both the development of an understanding of the practice of software engineering and an appreciation of the fact that evidence must be appropriate to its purpose and intended audience, should go some way towards closing the gap between empirical software engineering and software engineering practice.

References

[1] M.Shepperd, "Empirically-based Software Engineering", ***UP****GRADE*, IV (4), Aug. 2003, 37-41.

[2] M.V.Zelkowitz, D.R. Wallace, D.W. Binkley, 'Experimental validation of new software technology', in *Lecture Notes on Empirical Software Engineering*, N.Juristo, A.M. Moreno (eds.), World Scientific Publishing Co., 2002, pp. 229-263.

[3] D.E.Perry, A.A.Porter and L.G.Votta, "Empirical studies of software engineering: a roadmap", in *Proceedings of the International Conference on The Future of Software Engineering*, A. Finkelstein (ed.), ACM Press, 2000, pp.345-355.

[4] F. Lanubile, 'Empirical evaluation of software maintenance technologies', *Empirical Software Engineering*, 2, 1997, pp 97-108.

[5] C.Seaman, "Methods in empirical studies of software engineering', *IEEE Transactions on Software Engineering*, 25(4), 1999, pp.557-572.

[6] S.E.Sim, J.Singer and M-A Storey, "Beg, Borrow or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research", *Empirical Software Engineering*, 6, 2001, pp.85-93

[7] R.L. Glass, I. Vessey and V. Ramesh, "Research in software engineering: an analysis of the literature", *Information and Software Technology*, 44, 2002, pp.491-506.

[8] J. Munch, D.Rombach and I.Rus "Creating an advanced software engineering laboratory by combining empirical studies with process simulation", *Proceedings of Prosim 2003*, Portland, Oregon, 2003.

[9] B.Kitchenham, S.Linkman and J.Fry, "Experimenter induced distortions in empirical software engineering", *Proceedings of the 2nd workshop in the Workshop Series on Empirical Studies in Empirical Software Engineering*, Jedlitscha A and Ciolkowski M. (eds.), 2003, pp.7-15

[10] B.A.Kitchenham, S.L.Pfleeger, L.M.Pickard, P.W.Jones, D.C.Hoaglin, K. El Eman, J.Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering", *IEEE Transactions on Software Engineering*, 28(8), 2002, pp.721-734.

[11] J. Segal, "Some parallels between empirical software engineering and research in human-computer interaction", technical paper, *EASE & PPIG joint conference, University of Keele, UK, 2003*, 63-72.

[12] Y. Rogers, "New theoretical aproaches in HCI." To appear in *ARIST, Annual Review of Information Science and Technology*, 38, 2004.

[13] H.K.Klein and M.D.Myers, "A set of principles for conducting and evaluating interpretive field studies in information systems", *MIS Quarterly, 23(1)*, 1999, pp.67-93

[14] H.Robinson, J.Segal and H.Sharp, "The case for empirical studies of the practice of software development", *Proceedings of the 2nd workshop in the Workshop Series on Empirical Studies in Empirical Software Engineering*, Jedlitscha A and Ciolkowski M. (eds.), 2003, pp. 99-108.

[15] S.D.N. Cook and J.S. Brown "Bridging epistemologies: the generative dance between organizational knowledge and

organizational knowing", *Organization Science*, 10(4), 1999, pp. 381-400.

[16] A.F Chalmers. *What is this thing called science?* 2nd edition, Open University Press, Milton Keynes, UK, 1982.

[17] L. Bratthall and M. Jorgensen, 'Can you trust a single data source exploratory software engineering case study?', *Empirical Software Engineering*, 7, 2002, pp. 9 -26.

[18] J. Singer, T. Lethbridge, N. Vinson, N. Anquetil, 'An examination of software engineering work practices'. *Centre for Advanced Studies Conference (CASCON),* Toronto, Canada, 1997, pp. 1-15.

[19] W. Visser, J-M Hoc, 'Expert Software Design Strategies', in *Psychology of Programming*, J-M Hoc, T.R.G.Green, R. Samurcay, D. J. Gilmore (eds.), Academic Press, London, 1990, pp. 235-249.

[20] A.S.Lee, "A scientific methodology for MIS case studies", *MISQ*, 1989, pp.33-50

[21] Robinson H. and Sharp H., 'XP practice: why the twelve practices both are and are not the most significant thing', Proceedings of Agile Development Conference, IEEE Press, 2003, pp. 12-20.

[22] M. Cartwright, M. Shepperd, "An empirical investigation of an object-oriented software system", *IEEE Trans. On Softw. Eng*., 26(8), 2000, pp.786-796.

[23] B. Curtis, "Which comes first, the organisation or its processes?", *IEEE Software*, 15(6), 1998, pp. 10-13.

[24] T. Hall and D. Wilson, "Views of software quality: a field report", *IEE Proc-Softw.Eng*. 144(2), 1997, pp. 111-118.

[25] M. Morisio, M.Ezran and C.Tully, "Success and failure factors in software reuse", *IEEE Transactions on Software Engineering*, 28(4), 2002, pp.340-357.

[26] B. Boehm and R. Turner, *Balancing Agility and Discipline A Guide for the Perplexed*, Addison Wesley, 2004.

[27] P.E.Waterson, C.W.Clegg and C.M.Axtell, "The dynamics of work organization, knowledge and technology during software development", *Int. J. Human-Computer Studies*, 46, 1997, pp. 79-101.